

Alunos: Cecilia, Ivan e Paulo

Precisamos dividir a lista em outras sub-listas guardando as posições de início meio e fim como parâmetros de ordenação:

```
def mergesort(lista, inicio=0, fim=None)
    if fim is None:
        fim = len(lista)
```

None é um valor que você pode atribuir a uma variável que significa vazio, diferente do **pass** que não faz nada, ele consegue armazenar algo.

Então criamos as variáveis:

- **lista**, é onde são armazenados
- **início** indica a posição a estar na esquerda
- **meio** servindo para alocar a divisão
- **fim**

fim - início > 1

Vai dividir recursivamente o array até se tornar um subarray que possua somente 1 elemento.

Qual a função de um algoritmo recursivo?

A ideia básica consiste em diminuir sucessivamente o problema em um problema menor, até o tamanho que permita resolvê-lo de forma direta, sem recorrer a si mesmo.

Fazendo o processo para estar ordenado de menor para maior com apenas 1 array e combinar 2 subarrays e sucessivamente para depois ter 1 array.

```
if fim - inicio > 1:
    meio = (fim + inicio) // 2
    Classe.mergesort(lista, inicio, meio)
    Classe.mergesort(lista, meio, fim)
    Classe.merge(lista, inicio, meio, fim)
```

Classe.mergesort (e suas variáveis) realiza a recursividade e divide os elementos da lista.

def merge vai realizar a junção

left = lista[inicio:meio], indica seguir do início até o fim da lista e o **right = lista[meio:fim]** do meio até o fim da lista

```
def merge(lista, inicio, meio, fim):
    left = lista[inicio:meio]
    right = lista[meio:fim]
```

for k in range (inicio,fim)

Vai verificar se quem está no topo da lista da esquerda é menor de quem está no topo da lista da direita, então coloca o valor no lugar indicado do lado esquerdo, se não, adiciona na posição ao lado direito.

Código limpo:

```
for k in range(inicio, fim):
    if i >= len(left):
        lista[k] = right[j]
        j += 1
    elif j >= len(right):
        lista[k] = left[i]
        i += 1
    elif left[i] < right[j]:
        lista[k] = left[i]
        i += 1
    else:
        lista[k] = right[j]
        j += 1
```

Código com comentários:

```
for k in range(inicio, fim): # k é que verifica o topo da lista
    if i >= len(left): # se o topo da lista da esquerda for maior do que o
topo da lista da direita
        lista[k] = right[j] # adiciona na lista direita na posição
j(direita) da lista
        j += 1 # avança a posição do topo da lista da direita
    elif j >= len(right):
        lista[k] = left[i]
        i += 1
    elif left[i] < right[j]:# se o número na posição i (esquerdo) for
menor que da posição j (direito)
        lista[k] = left[i] # coloca o número na posição i (esquerdo)
        i += 1
    else:
        lista[k] = right[j] # adiciona o topo da lista na direita da
posição a direita
        j += 1
```

Código Completo:

```
def mergesort(lista, inicio=0, fim=None):  
    if fim is None:  
        fim = len(lista)  
  
    if fim - inicio > 1:  
        meio = (fim + inicio) // 2  
        Classe.mergesort(lista, inicio, meio)  
        Classe.mergesort(lista, meio, fim)  
        Classe.merge(lista, inicio, meio, fim)  
  
def merge(lista, inicio, meio, fim):  
    left = lista[inicio:meio]  
    right = lista[meio:fim]  
  
    i, j = 0, 0  
    for k in range(inicio, fim):  
        if i >= len(left):  
            lista[k] = right[j]  
            j += 1  
        elif j >= len(right):  
            lista[k] = left[i]  
            i += 1  
        elif left[i] < right[j]:  
            lista[k] = left[i]  
            i += 1  
        else:  
            lista[k] = right[j]  
            j += 1  
  
    return lista
```

Percebemos que sua Performance:

- No melhor caso (dados curtos): $O(N \log de N)$
- No pior caso (dados longos): $O(N \log de N)$

```
-----RELATÓRIO DE VELOCIDADE E EFICIÊNCIA-----

Sera analisado o tempo medio vetores
com os valores de 0 a 999

Vetores com 10
vetor_ordenado: 0.00027971 #tempo medio de criação do vetor
merge : 0.00006200 #tempo medio para o algoritmo organizar em segundos
shell : 0.00004255
quick : 0.00004298
bubble: 0.00005055
select: 0.00003112
insert: 0.00004698

Vetores com 100
vetor_ordenado 0.04396549
merge : 0.00036144
shell : 0.00087624
quick : 0.00040645
bubble: 0.00211883
select: 0.00117558
insert: 0.00086900

Vetores com 1000
vetor_ordenado 24.61177474
merge : 0.00478459
shell : 0.10300057
quick : 0.00510474
bubble: 0.30475405
select: 0.15221638
insert: 0.10891036
```

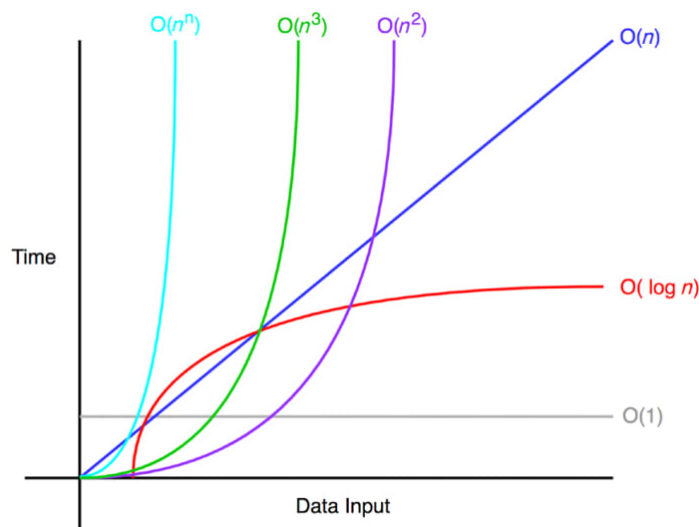
```
C:\terminal\comparacoes-algoritmos-ordenacao>py main.py
-----RELATÓRIO DE VELOCIDADE E EFICIÊNCIA-----

Sera analisado o tempo medio vetores
com os valores de 0 a 999

Vetores com 10
vetor_ordenado: 0.00013918 #tempo medio de criação do vetor
merge : 0.00003189 #tempo medio para o algoritmo organizar em segundos
shell : 0.00002099
quick : 0.00002803
bubble: 0.00003799
select: 0.00003997
insert: 0.00002602

Vetores com 100
vetor_ordenado 0.04306114
merge : 0.00043211
shell : 0.00153504
quick : 0.00051842
bubble: 0.00235282
select: 0.00176172
insert: 0.00105199

Vetores com 1000
vetor_ordenado 26.35210946
merge : 0.00526756
shell : 0.11345159
quick : 0.00556437
bubble: 0.21633080
select: 0.12561586
insert: 0.09282250
```



Qual a vantagem?

Um programa recursivo é mais elegante e menor que a sua versão iterativa e o mergesort é bem estável em diferentes situações.

Qual a desvantagem?

Ele é recursivo usa um vetor auxiliar durante a ordenação, ocasionando maior uso da memória.

Problema que isso pode gerar:

Se um algoritmo recursivo faz muitas chamadas, ele pode causar um estouro de pilha (stack overflow), ou seja, ficar sem memória suficiente para continuar a execução do programa.

Links utilizados:

<https://dev.to/b0nbon1/understanding-big-o-notation-with-javascript-25mc>

<https://stackoverflow.com/questions/47973242/what-is-the-difference-between-pass-and-none-in-python>

https://www.w3schools.com/python/ref_func_len.asp

[https://realpython.com/len-python-function/#:~:text=The%20function%20len\(\)%20is,of%20items%20in%20a%20list.](https://realpython.com/len-python-function/#:~:text=The%20function%20len()%20is,of%20items%20in%20a%20list.)

<https://www.youtube.com/watch?v=5prE6Mz8Vh0&t=684s>

<https://docs.python.org/pt-br/3/tutorial/controlflow.html>